يوسف خالد سيد

190203

Hill cipher:

Hill cipher:

from tkinter import \*

from math import gcd

root = Tk()

root.geometry("500x300")

root.title("Hill Cipher")

key\_label = Label(root, text="Enter the key:")

key\_label.pack()

key\_entry = Entry(root, width=50)

key\_entry.pack()

plain\_label = Label(root, text="Enter the plaintext:")

plain\_label.pack()

plain\_entry = Entry(root, width=50)

plain\_entry.pack()

cipher\_label = Label(root, text="Ciphertext:")

cipher\_label.pack()

cipher\_output = Label(root, text="")

cipher\_output.pack()

def generate\_key\_matrix(key\_str):

key\_nums = list(map(int, key\_str.split()))

n = int(len(key\_nums)\*\*0.5)

key\_matrix = [[0 for \_ in range(n)] for \_ in range(n)]

for i in range(n):

for j in range(n):

key\_matrix[i][j] = key\_nums[n\*i+j]

return key\_matrix

def encrypt(key\_matrix, plaintext):

n = len(key\_matrix)

plaintext = plaintext.upper().replace(" ", "")

plaintext += "X"\*(n-(len(plaintext)%n))

ciphertext = ""

for i in range(0, len(plaintext), n):

plain\_vec = [[ord(char)-65] for char in plaintext[i:i+n]]

cipher\_vec = [[0] for \_ in range(n)]

for j in range(n):

for k in range(n):

cipher\_vec[j][0] += key\_matrix[j][k] \* plain\_vec[k][0]

cipher\_vec[j][0] %= 26

for vec in cipher\_vec:

ciphertext += chr(vec[0]+65)

return ciphertext

def encrypt\_message():

key\_str = key\_entry.get()

plaintext = plain\_entry.get()

key\_matrix = generate\_key\_matrix(key\_str)

ciphertext = encrypt(key\_matrix, plaintext)

cipher\_output.config(text=ciphertext)

encrypt\_button = Button(root, text="Encrypt", command=encrypt\_message)

encrypt\_button.pack()

root.mainloop()

Row transposition:

from tkinter import \*

def encrypt():

# Retrieve the key and plaintext entered by the user

key = key\_entry.get()

plaintext = plaintext\_entry.get()

# Determine the number of columns in the transposition grid

num\_columns = len(key)

# Pad the plaintext with 'X' characters if necessary

if len(plaintext) % num\_columns != 0:

plaintext += 'X' \* (num\_columns - len(plaintext) % num\_columns)

# Create a list of strings, each representing a row in the transposition grid

rows = []

for i in range(0, len(plaintext), num\_columns):

rows.append(plaintext[i:i+num\_columns])

# Create a dictionary mapping each column index to its corresponding key character

key\_dict = {}

for i, c in enumerate(key):

key\_dict[i+1] = c

# Sort the column indexes in ascending order based on the key characters

sorted\_columns = [k for k, v in sorted(key\_dict.items(), key=lambda item: item[1])]

# Build the ciphertext by reading off the rows in the transposition grid in the sorted order

ciphertext = ''

for col in sorted\_columns:

for row in rows:

ciphertext += row[col-1]

# Display the ciphertext in the output label

ciphertext\_output.config(text=ciphertext)

# Create the tkinter window

root = Tk()

root.title('Row Transposition Cipher')

root.geometry('500x300')

# Add input fields for the key and plaintext

key\_label = Label(root, text='Key:')

key\_label.pack()

key\_entry = Entry(root, width=50)

key\_entry.pack()

plaintext\_label = Label(root, text='Plaintext:')

plaintext\_label.pack()

plaintext\_entry = Entry(root, width=50)

plaintext\_entry.pack()

# Add a button to trigger the encryption process

encrypt\_button = Button(root, text='Encrypt', command=encrypt)

encrypt\_button.pack()

# Add a label to display the resulting ciphertext

ciphertext\_label = Label(root, text='Ciphertext:')

ciphertext\_label.pack()

ciphertext\_output = Label(root, text='')

ciphertext\_output.pack()

# Start the tkinter event loop

root.mainloop()

Monoalphabitic cipher:

from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QTextEdit, QPushButton, QVBoxLayout

from PyQt5.QtGui import QFont

# define the Monoalphabetic cipher function

def monoalphabetic\_cipher(text, key, mode):

alphabet = "abcdefghijklmnopqrstuvwxyz"

key = key.lower()

key\_dict = {}

for i in range(len(key)):

key\_dict[alphabet[i]] = key[i]

if mode == "encrypt":

result = ""

for char in text:

if char.isalpha():

if char.isupper():

result += key\_dict[char.lower()].upper()

else:

result += key\_dict[char]

else:

result += char

return result

elif mode == "decrypt":

result = ""

for char in text:

if char.isalpha():

if char.isupper():

result += list(key\_dict.keys())[list(key\_dict.values()).index(char.lower())].upper()

else:

result += key\_dict[char]

else:

result += char

return result

class CipherApp(QWidget):

def \_init\_(self):

super().\_init\_()

self.initUI()

def initUI(self):

# set the window title and dimensions

self.setWindowTitle('Monoalphabetic Cipher')

self.setGeometry(100, 100, 400, 400)

# create the label and text input for the key

key\_label = QLabel('Enter the key:')

key\_input = QLineEdit()

# create the label and text input for the plaintext/ciphertext

text\_label = QLabel('Enter the text:')

text\_input = QTextEdit()

# create the labels to display the result of the encryption/decryption

result\_label = QLabel('Result:')

self.result\_output = QTextEdit()

self.result\_output.setReadOnly(True)

# create the encryption and decryption buttons

encrypt\_button = QPushButton('Encrypt', self)

encrypt\_button.clicked.connect(lambda: self.encrypt(key\_input.text(), text\_input.toPlainText()))

decrypt\_button = QPushButton('Decrypt', self)

decrypt\_button.clicked.connect(lambda: self.decrypt(key\_input.text(), text\_input.toPlainText()))

# create the vertical layout and add all the widgets to it

vbox = QVBoxLayout()

vbox.addWidget(key\_label)

vbox.addWidget(key\_input)

vbox.addWidget(text\_label)

vbox.addWidget(text\_input)

vbox.addWidget(result\_label)

vbox.addWidget(self.result\_output)

vbox.addWidget(encrypt\_button)

vbox.addWidget(decrypt\_button)

# set the font for all the widgets

font = QFont('Arial', 12)

key\_label.setFont(font)

key\_input.setFont(font)

text\_label.setFont(font)

text\_input.setFont(font)

result\_label.setFont(font)

self.result\_output.setFont(font)

encrypt\_button.setFont(font)

decrypt\_button.setFont(font)

# set the layout for the window

self.setLayout(vbox)

def encrypt(self, key, text):

# call the Monoalphabetic cipher function to encrypt the text

ciphertext = monoalphabetic\_cipher(text, key, "encrypt")

self.result\_output.setPlainText(ciphertext)

def decrypt(self, key, text):

# call the Monoalphabetic cipher function to decrypt the text

plaintext = monoalphabetic\_cipher(text, key, "decrypt")

     self.result

Rail fence cipher:

def encryptRailFence(text, key):

rail = [['\n' for i in range(len(text))]

for j in range(key)]

dir\_down = False

row, col = 0, 0

for i in range(len(text)):

if (row == 0) or (row == key - 1):

dir\_down = not dir\_down

rail[row][col] = text[i]

col += 1

if dir\_down:

row += 1

else:

row -= 1

result = []

for i in range(key):

for j in range(len(text)):

if rail[i][j] != '\n':

result.append(rail[i][j])

return("" . join(result))

def decryptRailFence(cipher, key):

rail = [['\n' for i in range(len(cipher))]

for j in range(key)]

# to find the direction

dir\_down = None

row, col = 0, 0

# mark the places with '\*'

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key - 1:

dir\_down = False

# place the marker

rail[row][col] = '\*'

col += 1

# find the next row

# using direction flag

if dir\_down:

row += 1

else:

row -= 1

index = 0

for i in range(key):

for j in range(len(cipher)):

if ((rail[i][j] == '\*') and

(index < len(cipher))):

rail[i][j] = cipher[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(cipher)):

# check the direction of flow

if row == 0:

dir\_down = True

if row == key-1:

dir\_down = False

# place the marker

if (rail[row][col] != '\*'):

result.append(rail[row][col])

col += 1

# find the next row using

# direction flag

if dir\_down:

row += 1

else:

row -= 1

return("".join(result))

# Example usage

plaintext = "youssef khaled "

key = 3

def encryptRailFence(text, key):

rail = [['\n' for i in range(len(text))]

for j in range(key)]

dir\_down = False

row, col = 0, 0

for i in range(len(text)):

if (row == 0) or (row == key - 1):

dir\_down = not dir\_down

rail[row][col] = text[i]

col += 1

if dir\_down:

row += 1

else:

row -= 1

result = []

for i in range(key):

for j in range(len(text)):

if rail[i][j] != '\n':

result.append(rail[i][j])

return("" . join(result))

def decryptRailFence(cipher, key):

rail = [['\n' for i in range(len(cipher))]

for j in range(key)]

# to find the direction

dir\_down = None

row, col = 0, 0

# mark the places with '\*'

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key - 1:

dir\_down = False

# place the marker

rail[row][col] = '\*'

col += 1

# find the next row

# using direction flag

if dir\_down:

row += 1

else:

row -= 1

index = 0

for i in range(key):

for j in range(len(cipher)):

if ((rail[i][j] == '\*') and

(index < len(cipher))):

rail[i][j] = cipher[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(cipher)):

# check the direction of flow

if row == 0:

dir\_down = True

if row == key-1:

dir\_down = False

# place the marker

if (rail[row][col] != '\*'):

result.append(rail[row][col])

col += 1

# find the next row using

# direction flag

if dir\_down:

row += 1

else:

row -= 1

return("".join(result))

# Example usage

plaintext = "Youssef khaled"

key = 3

# Encrypt the plaintext using the Rail Fence Cipher

ciphertext = encryptRailFence(plaintext, key)

print("Encrypted message:", ciphertext)

# Decrypt the ciphertext using the Rail Fence Cipher

decrypted\_plaintext = decryptRailFence(ciphertext, key)

print("Decrypted message:", decrypted\_plaintext)

# Encrypt the plaintext using the Rail Fence Cipher

ciphertext = encryptRailFence(plaintext, key)

print("Encrypted message:", ciphertext)

# Decrypt the ciphertext using the Rail Fence Cipher

decrypted\_plaintext = decryptRailFence(ciphertext, key)

print("Decrypted message:", decrypted\_plaintext)

play faire cipher:

from tkinter import \*

# Playfair cipher functions

def playfair\_cipher(text, key, mode):

# Create the Playfair square

key = key.replace(" ", "").upper()

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZY"

playfair\_square = ""

for letter in key:

if letter not in playfair\_square:

playfair\_square += letter

for letter in alphabet:

if letter not in playfair\_square:

playfair\_square += letter

playfair\_square = [playfair\_square[i:i+5] for i in range(0, 25, 5)]

# Encrypt or decrypt the text

result = ""

for i in range(0, len(text)-1, 2):

pair = text[i:i+2].upper()

if pair[0] == pair[1]:

pair = pair[0] + "X"

row1, col1 = divmod(playfair\_square.index(pair[0]), 5)

row2, col2 = divmod(playfair\_square.index(pair[1]), 5)

if row1 == row2:

if mode == "encrypt":

col1 = (col1 + 1) % 5

col2 = (col2 + 1) % 5

else:

col1 = (col1 - 1) % 5

col2 = (col2 - 1) % 5

elif col1 == col2:

if mode == "encrypt":

row1 = (row1 + 1) % 5

row2 = (row2 + 1) % 5

else:

row1 = (row1 - 1) % 5

row2 = (row2 - 1) % 5

else:

col1, col2 = col2, col1

result += playfair\_square[row1][col1] + playfair\_square[row2][col2]

return result

def encrypt():

text = plaintext\_entry.get()

key = key\_entry.get()

ciphertext = playfair\_cipher(text, key, "encrypt")

ciphertext\_entry.delete(0, END)

ciphertext\_entry.insert(0, ciphertext)

def decrypt():

text = ciphertext\_entry.get()

key = key\_entry.get()

plaintext = playfair\_cipher(text, key, "decrypt")

plaintext\_entry.delete(0, END)

plaintext\_entry.insert(0, plaintext)

# GUI setup

root = Tk()

root.title("Playfair Cipher")

# Create labels and entry boxes

plaintext\_label = Label(root, text="Plaintext:")

plaintext\_label.grid(row=0, column=0, sticky=W)

plaintext\_entry = Entry(root)

plaintext\_entry.grid(row=0, column=1)

key\_label = Label(root, text="Key:")

key\_label.grid(row=1, column=0, sticky=W)

key\_entry = Entry(root)

key\_entry.grid(row=1, column=1)

ciphertext\_label = Label(root, text="Ciphertext:")

ciphertext\_label.grid(row=2, column=0, sticky=W)

ciphertext\_entry = Entry(root)

ciphertext\_entry.grid(row=2, column=1)

# Create encrypt and decrypt buttons

encrypt\_button = Button(root, text="Encrypt", command=encrypt)

encrypt\_button.grid(row=3, column=0)

decrypt\_button = Button(root, text="Decrypt", command=decrypt)

decrypt\_button.grid(row=3, column=1)

root.mainloop()